Part 2: Basics of Gaussian Processes and Expected Improvement

December 21, 2020

Hey everyone! Welcome to the first *ZACLAB* post related to data science, optimization, and statistical methods. Today we will be talking about Gaussian Processes (GPs) and Expected Improvement (EI) as it relates to Bayesian Optimization (BO). I will provide code throughout the post. In fact, the majority of the material here will be in reference to code in Python that will be posted.

1 Gaussian Processes

Before we do anything related to BO, we have to build our own GP. A GP is a statistical way of doing regression. By statistical, I mean that by training a GP you not only get a mean but a standard deviation as well...pretty fancy! We will look at single output $\hat{\mu} \pm \hat{\sigma}$ GPs with training data X and Y with dimensionality p. There is **a lot** more to be said about GPs than this, but for now here are the two most important equations for GPs to understand:

$$\hat{\mu} = K(x, X)(K(X, X) + \sigma^2 I)^{-1} Y \tag{1}$$

$$\hat{\sigma}^2 = K(x, x) - K(x, X)(K(X, X)^{-1} K(X, x) \tag{2}$$

Next we must understand that K object. It's a matrix called a kernel. It is a mathematical relationship, in this case between two stationary points, say x_i and x_j with something like the squared exponential kernel... $K(x_i, x_j) = \sigma_f^2 exp(-r/2 * l^2)$ where $r = ||x_i - x_j||_2^2$ (or the squared L2 norm of the difference between x_i and x_j.

So let's apply the GP to a toy problem... $f(x) = x_1 - x_2 + U(0, 0.1)$ as shown in Figure 1. The black crosses is the data X associated with a given Y we will be working with. To apply the GP, calculate $\hat{\mu}$ for any point x_1, x_2 you want using the training data X, Y; that is, calculate $K(X, X)$ for your data and calculate $K(x, X)$ for any x you want to predict $\hat{\mu}$ and $\hat{\sigma}$.

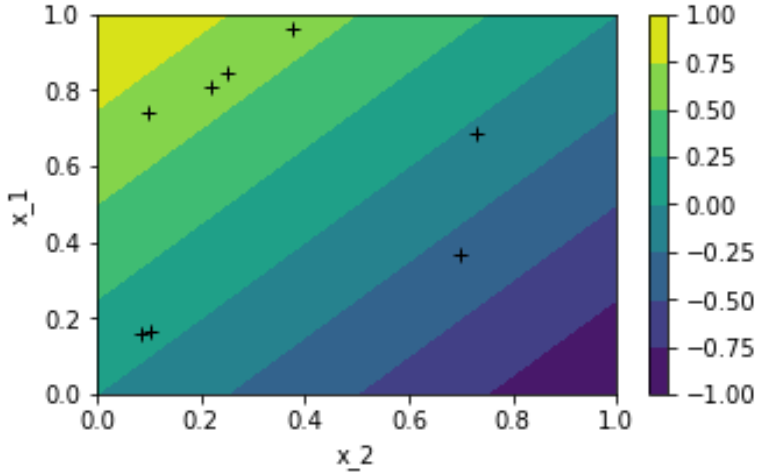


Figure 1: Toy Function

2 Hyperparameter Optimization for GPs

Something I forgot (intentionally) to mention in Section 1 was that GPs, while not having structure such as in linear or quadratic models, have hyperparameters. These are parameters that control how the GP behaves. In our example, the squared exponential kernel has three hyperparameters: a model error parameter σ, an amplitude parameter σ_f, and a length scale parameter l. We can define a likelihood function $L(\sigma, \sigma_f, l)$ based on these parameters.

$$L(\sigma, \sigma_f, l) = -0.5 Y K_y Y^T - 0.5 log(det(K_y)) - 0.5 n log(2\pi) \tag{3}$$

$$K_y = (K(X, X, \sigma_f, l) + \sigma^2 I)^{-1} \tag{4}$$

We can plot slices of L for the parameters in our problem as shown in Figure 2. We will talk about three different ways to get the optimal hyperparameters for this GP, (1) random search, (2) gradient descent method, and (3) BFGS method.

2.1 Random Search

The simplest way of getting optimal hyperparameters is to randomly search for them. This one is simple because it does not rely on gradients. I just calculate L for $M = 100$ random hyperparameters $[0.1, 20]$ and use the smallest L result.

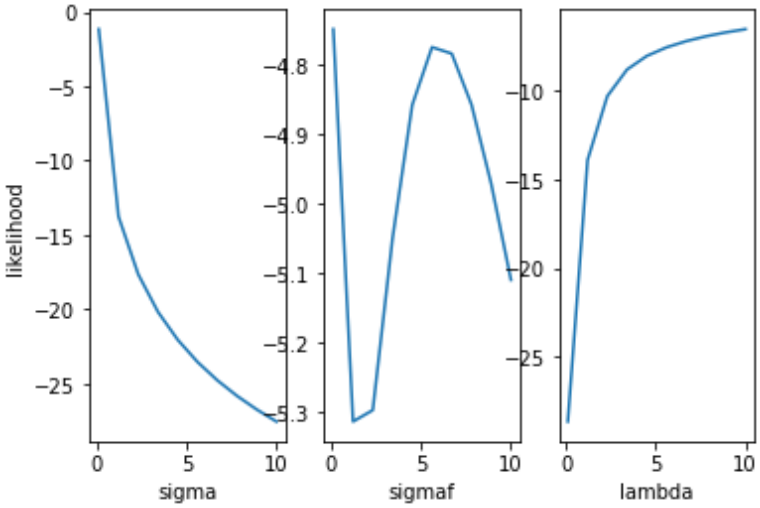


Figure 2: Likelihoods for Various Hyperparameters

2.2 Gradient Descent Method

Next we will use gradient information to improve search. We'd do this if the number of hyperparameters is large. The gradient of the likelihood function L is as follows:

$$\nabla L = (\nabla L_\sigma, \nabla L_{\sigma_f}, \nabla L_l) = 0.5Tr(\alpha\alpha^T - K_y^{-1}\nabla_i L) \tag{5}$$

$$\nabla_\sigma L = 2\sigma I \tag{6}$$

$$\nabla_{\sigma_f} L = 2/\sigma_f K(X, X) \tag{7}$$

$$\nabla_l L_{i,j} = K(x_i, x_j) r(x_i, x_j)/l^3 \tag{8}$$

$$\alpha = K_y^{-1} Y \tag{9}$$

The update scheme for gradient descent will be $\theta_{k+1} = \theta_k + \eta\nabla L$ where $\eta = 0.1$ for hyperparameters θ. Using a random initialization of the hyperparameters and $M = 1000$ iterations of gradient descent, we get the results shown in Figure 2(b). For those of you familiar with gradient descent, we use $-\nabla L$ rather than ∇L for our optimal step because we are maximizing L rather than minimizing. Note also that our hyperparameters θ must be > 0 so we modify our gradient descent to be *projected* gradient descent. Simply, if a constraint is violated we project the

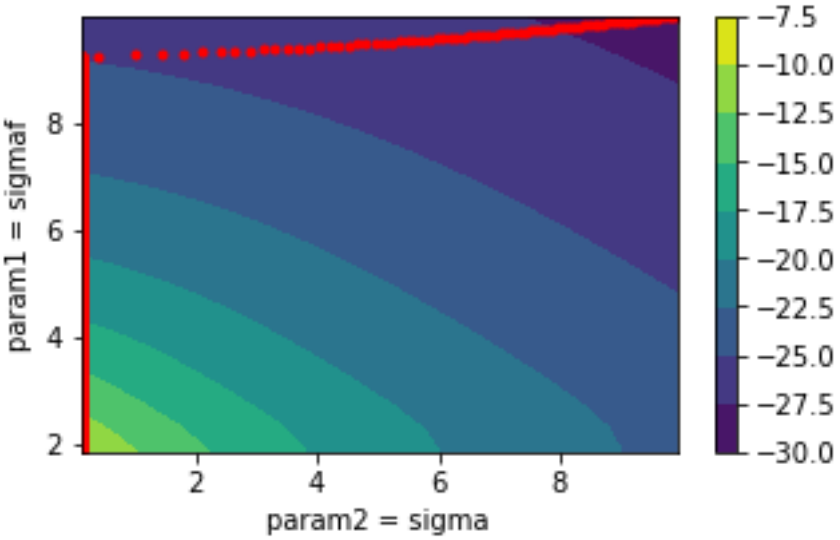


Figure 3: Gradient Descent *in Action!*

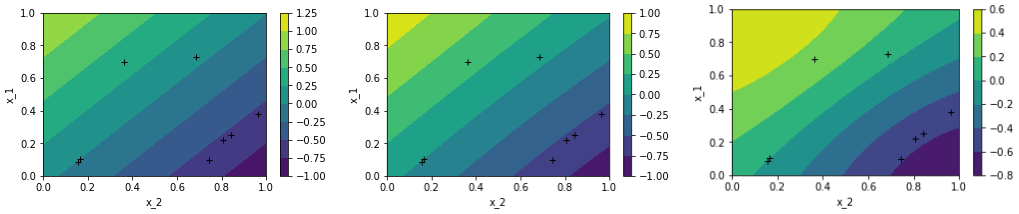
violated parameter onto the viable set Θ; in this case $\theta = 0$. An example is shown in Figure 3.

2.3 BFGS Method

The final thing I'd like to talk about with hyperparameter optimization is using the BFGS gradient method to solve the optimization problem. BFGS stands for *Broyden–Fletcher–Goldfarb–Shanno* and approximates the Hessian $\nabla^2 L$ of the objective function to speed up optimization. Here we will implement BFGS using *sci-py*'s *scipy.optimize.minimize* program. Just make sure to pass X and Y as constants to the optimizer so L and ∇L can be calculated, which can be done (and is done in the code) using wrapper functions. Additionally, just like projecting the failed gradients onto the feasible space Θ, we must tell *scipy.optimize.minimize* we have upper and lower bounds.

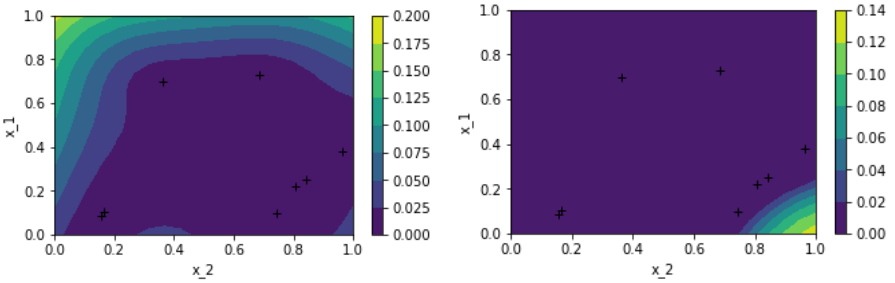
3 Expected Improvement

Next we implement a useful tool for optimization. It a function called *Expected Improvement* (EI), which is a way to optimize a random objective function mean $\hat{\mu}$ with uncertainty $\hat{\sigma}^2$. We will use it to design intelligent experiments, which will be a common theme in many of these posts. For example, if we have chemicals in concentrations X and cost Y for each combination, we will solve the problem:



(a) Random (b) Gradient Descent (c) BFGS

Figure 4: Results for Hyperparameter Optimization Methods



(a) Standard Deviation (b) Expected Improvement

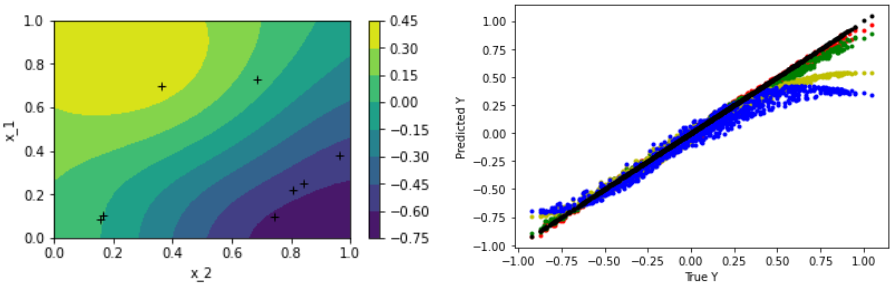
$$X^* = argmin\, \mu(X) \tag{10}$$

Where experimental data has been collected and modeled using a GP. As before, without going into too much detail, here is the basic equation for single-output EI where a single set of inputs is suggested ($q = 1$). Note that Φ and ϕ is the CDF and PDF of the standard normal distribution respectively.

$$EI(X) = (Y_{min} - \hat{\mu})\Phi(z) - \hat{\sigma}\phi(z) \tag{11}$$

$$z = (Y_{min} - \hat{\mu})/\hat{\sigma} \tag{12}$$

In the figures below we show a contour plot of some experiments with GP model standard deviations and it's respective EI plot. Note that in the minimization setting, the best region is on the lower right, but there is uncertainty in both lower right and upper left. The EI distinguishes the two and biases towards the lower right **as wanted**. This is valuable in expensive optimization processes where the question of which parameters to evaluate next is non-trivial and/or expensive.



(a) *Botorch* GP (b) Regressive Plots

Figure 5: Model Comparison and *Botorch*

4 EI and *Botorch*

We end this post with a discussion (and code posted) about *Botorch*. This is a robust Python implementation of a lot of what we have talked about. I have just written simple code to create a GP similar to what we did above, and optimize it's parameters using BFGS. In Figure 5(a) the contour looks similar to ours, and in Figure 5(b) the regressive predictions for $M = 1000$ random predictions on the toy problem largely adhere to the 45° line in black.

We this was fun! Next week we'll be talking about radial basis functions!